MODBUS_TCPIP - Driver of MODBUS_TCP/IP Protocol for OPEN MODBUS/TCP Mode User's Manual

Doc. No. ENP4036
Version: 29-08-2005

ASKOM® and **asix** ® are registered trademarks of ASKOM Spółka z o.o., Gliwice. Other brand names, trademarks, and registered trademarks are the property of their respective holders.

All rights reserved including the right of reproduction in whole or in part in any form. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without prior written permission from the ASKOM.

ASKOM sp. z o. o. shall not be liable for any damages arising out of the use of information included in the publication content.

Copyright © 2005, ASKOM Sp. z o. o., Gliwice

1. MODBUS_TCPIP - Driver of MODBUS_TCP/IP Protocol for OPEN MODBUS/TCP Mode

1.1. Driver Use

The driver MODBUS_TCPIP is designed for data exchange between the **asix** system and other computers/devices with use of the MODBUS protocol realized via an Ethernet network with the TCP/IP protocol. The default operating mode of the MODBUS_TCPIP driver is the Open Modbus/TCP mode, developed on the base of specification titled: *"OPEN MODBUS/TCP Specification" Release 1.0, issued 29.03.1999 by the firm Schneider Electric*. The driver allows simultaneous operation in both SLAVE and MASTER modes.

SLAVE Mode

The SLAVE mode consists in executing the read/write commands for the ASMEN data sent from other computers/devices, which are defined as MASTER in the MODBUS network. It is allowed to connect many computers operating as MASTER in the netwok.

The ASMEN variables are accessed in the SLAVE mode as MODBUS variables from one of the types specified below:

- CS (coil status),
- HR (holding registers),
- IR (input registers).

Declarations of assignment (mapping) of the ASMEN variables to the MODBUS variables are placed by the application designer in text files, which are read by the driver during starting the **asix** system (see: an example on the end of the chapter).

To read the ASMEN variables the following functions of the MODBUS protocol are implemented:

- Read Coil Status (function 01),
- Read Holding Registers (function 03),
- Read Input Registers (function 04).

To write the ASMEN variables the following functions of the MODBUS protocol are used:

- Preset Single Coil (function 05),
- Preset SingleRegister (function 06),
- Preset Multiple Coils (function 15),
- Preset Multiple Registers (function 16).

1.2. Declaration of Transmission Channel

The declaration of transmission channel for the SLAVE mode is as follows:

channel_name=MODBUS_TCPIP, SLAVE, port [, number]

where:

MODBUS_TCPIP - driver name;
SLAVE - operation in the SLAVE mode;
port - port no. where MASTER computers, which want to connect to the **asix** system, will be listened;
number - number assigned to an **asix** system computer in the MODBUS network (by default, 1).

NOTE
It is allowed to declare only one transmission channel executing the SLAVE mode.

1.3. SLAVE Driver Configuration

The driver configuration in the **asix** system is performed by means of the separate section named **[MODBUS_TCPIP_SLAVE]** in the application INI file. By using this section it is possible to declare:

- list of clients accepted by the driver;
- mapping the MODBUS variables to the ASMEN variables for individual clients;
- transfer method of the status for individual clients;
- method of refreshing the variables.

☑ ***PROTOCOL_TYPE = <IP_address>, OVATION***

Meaning - the support of stations operating according to the specification MODBUS RTU requires to use an item.
Parameter:
IP_address - address of the station operating as the master.

☑ ***CLIENT_IP_ADDRESS=client_IP_address [,READ_ONLY]***

Meaning - to the **asix** system only the clients which have an access permission nmay connect. The clients are declared by CLIENT_IP_ADDRESS items placed in the MODBUS_TCP_SLAVE section.
Default value - by default, each client may read and write **asix** system variables. It is possible to limit clients access only for reading the variables by using the READ_ONLY option. In this situation an attempt to write is ackowledged by sending a response telegram of the *exception* type with a code 1 which signifies an illegal function.
Parameter:
client_IP_address - IP address of the client accepted by the driver;
READ_ONLY - option allowing the client to read data only.

Number of client declarations not limited.

EXAMPLE

A declaration of the client with IP address 10.10.10.84 with unrestricted access to the **asix** system:

CLIENT_IP_ADDRESS=10.10.10.84

CONNECTION_TIMEOUT=client_IP_address,number

Meaning - the maximal time, which may elapse between successive queries from client (connection timeout), is declared for each connected client. After exceeding the connection timeout, the connection with the client is broken. In case of clients, for which the timeout declaration of is not defined, a default value of 5 minutes is assumed.

Parameter:

client_IP_address - IP address of a client,
number - timeout in minutes

EXAMPLE

The declaration of 10-minute connection timeout for the client with the address IP 10.10.10.84:

CONNECTION_TIMEOUT=10.10.10.84, 10

Transferring Status of Variables in Separate Registers and Coils

REGISTER_STATUS=client_IP_address

COIL_STATUS=client_IP_address

Meaning - the MODBUS protocol does not use a status with reference to transferred values of registers and coils. For this reason the status may be joined to the standard MODBUS transfer only in an artificial way. The driver under consideration may transfer the **asix** variable status by placing it in the next element following one which contains the variable value. For registers it is the next register, for coils – the next coil. In case of registers the space for status is sufficient (16 bits), for coils the status must be limited to two states (0 - good , 1 - bad).

By example, if the variable value is transferred as the register 10, then the status is transferred as the register 11. If the coil state is transferred in the bit no. 5, then the coil status is transferred in the bit no. 6.

The applied method allows to send the variable value and its status in the same telegram which ensures the data compactness.

The mode of status transferring is declared individually for each client, separately for registers and for coils. Declarations enable:

- transferring status of registers,

- transferring status of coils.

Default value - by default, the status is not transferred either for registers or for coils.

Parameter:

REGISTER_STATUS - transferring status of register,
COIL_STATUS - transferring status of coils,
client_IP_address - client IP address.

Transferring Status of Variables Without Using Separate Registers or Coils

REGISTER_STATUS_ERROR=client_IP_address,number

Meaning - in case of work without transferring statuses in separate registers or coils, a problem of transfering the information that the variable value is invalid (e.g. errors of communication with data source) arises. The convention is assumed, according to which it is possible to declare the value transferred in case of incorrect status for registers (by default, 0xffff).

Parameter:

client_IP_address - client IP address;
number - 16-bit value (HEX), transferred in case of an incorrect variable status.

EXAMPLE

The declaration of transfering the number 0x8000 to the client 10.10.10.84 in case of an error signalized in the status of the variable mapped on MODBUS registers:

REGISTER_STATUS_ERROR=10.10.10.84, 0x8000

NOTE
In case of coils, because of a binary character of a variable, there is no possibility to transfer the information about the status in such way that it might differ from the correct variable value. Considering it you should also be assumed as a rule that in the work mode without status transferring the registers should be used.

Declaration of Mapping MODBUS Variables to the ASMEN Variables

Because the MODBUS protocol does not use any names of variables but numbers of registers and coils, it is necessary to prepare configuration files for mapping MODBUS variables to ASMEN variables.

The declaration of mapping a MODBUS variable to an ASMEN variable is as follows:

asix_name, MODBUS_address [, credibility_time]

where:

asix_name - process variable name of the **asix** system; it must have its equivalent amongst names of process variables in ASMEN files;
MODBUS_address - type and index of a MODBUS variable, by means of which the process variable value is accessed; depending on the MODBUS variable type, its value is transferred as one bit, one register (a

variable of SHORT or USHORT type) or two successive registers (a variable of type FLOAT, LONG or ULONG);

validity_time - time interval (in seconds), in which the variable value is assumed to be correct. This time is defined as a difference between the time stamp of a query from client and time stamp of an actually accessed variable value. By default, it is equal to 5.

An accepted format of the MODBUS address is as follows:

HR<reg_no> or IR<reg_no> or CS<coil_no>

where:

reg_no - number of register HR or IR;
coil_no - number of coil.

The number of MODBUS registers used for transfering the variable value is closely related to the ASMEN variable type:

- for WORD type variables it is one register or one coil;
- for INT16 type variables it is one register;
- for DWORD, LONG or FLOAT type variables it is two registers;

or in case of status transfer:

- for WORD type variables it is two registers or two coils;
- for INT16 type variables it is two registers;
- for DWORD, LONG or FLOAT type variables it is two registers.

The method of register content interpretation for FLOAT, LONG and ULONG type numbers of MODBUS:

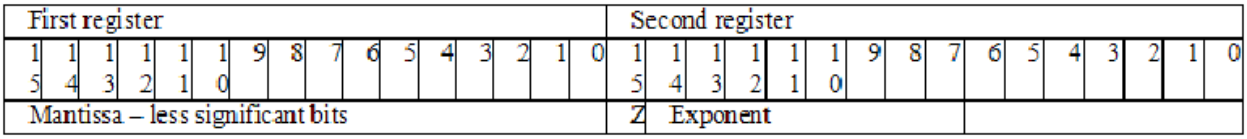
First register
Second register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Mantissa – less significant bits
S Exponent

Figure 1. Format FLOAT (IEEE 745).

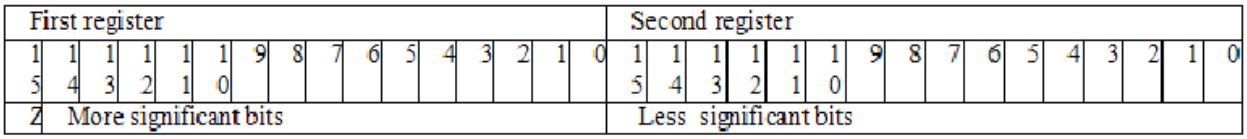
First register
Second register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
S More significant bits
Less significant bits

Figure 2. Format LONG.

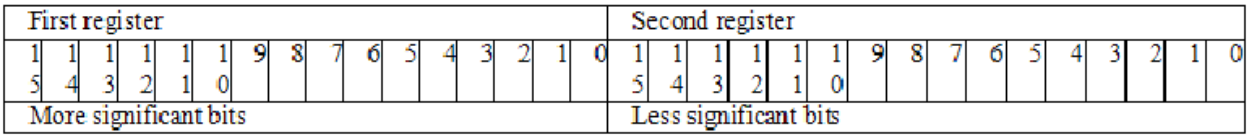
First register
Second register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
More significant bits
Less significant bits

Figure 3. Format ULONG.

EXAMPLE

In ASMEN the variables X_WORD, X_INT, X_DWORD, X_LONG, X_FLOAT are defined as follows:

X_WORD, ED120.2, CHAN1, 1, 1, NOTHING
X_INT, ED130.2, CHAN1, 1, 1, NOTHING_INT
X_DWORD, EL140.2, CHAN1, 1, 1, NOTHING_DW
X_LONG, EL150.2, CHAN1, 1, 1, NOTHING_LONG

X_FLOAT, EG160.2, CHAN1, 1, 1, NOTHING_FP

Mapping the values of these variables to a continuous area of registers beginning from the register HR1 (without status transferring) is as follows:
X_WORD, HR1
X_INT, HR2
X_DWORD, HR3
X_LONG, HR5
X_FLOAT, HR7

Mapping the values of these variables to a continuous area of registers beginning from the register HR1 (with status transferring) is as follows:
X_WORD, HR1
X_INT, HR3
X_DWORD, HR5
X_LONG, HR8
X_FLOAT, HR11

Declarations of mappings of MODBUS variables to ASMEN variables are placed in text files. Localization of files with d mapping eclarations is specified by means of the item:

MAP_FILE=client_IP_address,file_name

Meaning - localization of files with declarations of mappings.
Parameter:
client_IP_address - client IP address;
file name - name of a file containing declaration of mappings.

The purpose of connection of a file with a client IP address is to enable individual clients an independent way of mapping.

The number of items with declarations of mapping files is unlimited.

Refreshing Variables Accessed by the Driver

The driver may use one of two strategies of handling the variables exported from the **asix** system.

The strategy I (default) assumes that all the variables, the names of which were declared in the file with mapping declarations, are periodically read by a separate driver thread from the ASMEN cache and placed in a common accessible driver buffer. Threads supporting connection with clients build response telegrams on the basis of this buffer content, without necessity to access to the ASMEN API.

The period of reading from the ASMEN cache to shared driver buffer is configured by means of the item:

DATA_BUFFER_UPDATE_PERIOD=number

Meaning - enables declaring the period of reading from the ASMEN cache to shared driver buffer.
Default value - by default the period of reading is equal to 1 second.
Parameter:

number - period of reading from the ASMEN cache in seconds.

The strategy II (option) assumes that each thread takes data from the ASMEN cache individually in response to the queries sent by the client. As a result the thread reads from the ASMEN cache only these data, which are required by the client at that moment.

The strategy II should be also activated when **asix** uses the driver only to import data from an other system. In such mode periodical refreshing of contents of driver buffers has no sense.

The strategy II is activated by means of the item:

☑ ***READ_ON_REQUEST = YES***

Values of variables, which are in the ASMEN cache in the moment of executing the read command, may be out-of-date because in the period preceding the client query the refreshing of variables under consideration might be inactive (the **asix** system was just run or none of ASMEN clients was interested in refreshing these variables). Therefore the thread preparing a transfer for the client checks the time stamp actually accessed variables values. If the time stamp value is not contained in the range of *credibility_time* specified in the variable mapping declaration, then the thread reads the variable value directly from the driver and just then returns the answer to the client.

By default, the driver realizes the strategy I with a frequency of updating the content of common accessible buffer equal to 1 second.

☑ ***LOG_FILE=file_name***

Meaning - the item allows to define a file where all diagnostic messages of the driver in the SLAVE mode and the information about contents of telegrams received/sent by the driver in this mode are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value - by default, the log file is not created.
file name - log file name.

☑ ***LOG_OF_TELEGRAMS =YES|NO***

Meaning - the item allows to write to the log file the contents of telegrams sent between the driver working in the SLAVE mode and network clients. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value - by default, the driver does not write the contents of telegrams to the log file.

☑ ***LOG_FILE_SIZE=number***

Meaning - the item allows to specify the log file size.
Default value - by default, the item assumes that the log file has a size of 1 MB.
Parameter:

number - log file size in MB.

MASTER Mode

The MASTER mode is the MASTER function implementation in the MODBUS network protocol (in RTU mode) based on ETHERNET with the TCP/IP protocol.

The MASTER mode has implemented the following data types:

CS	- Coil Statuses;
IS	- Input Statuses;
HR	- Holding Registers;
IR	- Input Registers;
HRL	- 2 successive Holding Registers treated as a double word in INTEL format;
HRF	- 2 successive Holding Registers treated as a floating-point number in INTEL format;
HRLM	- 2 successive Holding Registers treated as a double word in MOTOROLA format;
HRFM	- 2 successive Holding Registers treated as a floating-point number in MOTOROLA format;
IRL	- 2 successive Input Registers treated as a double word in INTEL format;
IRF	- 2 successive Input Registers treated as a floating-point number in INTEL format;
IRLM	- 2 successive Input Registers treated as a double word in MOTOROLA format;
IRFM	- 2 successive Input Registers treated as floating-point number in MOTOROLA format.

EXAMPLES

CS22	- Coil 22
IS197	- Input 197
HR118	- Holding Register 118
IR25	- Input Register 25

and the following functions of the MODBUS protocol:

Read Coil Statuses	(function 01),
Read Input Statuses	(function 02),
Read Holding Registers	(function 03),
Read Input Registers	(function 04),
Preset Single Coil	(function 05),
Preset SingleRegister	(function 06),
Preset Multiple Registers	(function 16 limited to writing a pair of registers).

The MODBUS driver is loaded as a DLL automatically.

1.4. Declaration of Transmission Channel

The full syntax of declaration of transmission channel for the MODBUS_TCPIP driver channel in the MASTER mode is as follows:

channel_name=MODBUS_TCPIP, MASTER, port,IP_address [, number [, binary [, register]]]

where:

MODBUS_TCPIP	- driver name;
MASTER	- MASTER mode;
port	- number of the port by means of which the connection with a slave type device numbered *number* will be executed;
IP_address	- IP address of the device;
number	- number of the slave type device supported in this channel (by default, 1);
binary	- maximal number of binary signals in one query (by default, 32*8);
register	- maximal number of registers in one query (by default, 127).

For each slave type device supported in the MASTER mode a separate declaration of transmission channel is required.

1.5. MASTER Mode Configuration

The configuring the the MASTER mode is executed by means of the separate section named **[MODBUS_TCPIP_MASTER]**. By means of this section it is possible to declare:

- time to establish a connection on the startup stage,
- timeout of waiting for an answer from the slave type device,
- log file and its size,
- log of telegrams.

☑ ***PROTOCOL_TYPE = <IP_address>, OVATION***

Meaning - the support of stations working according to the specification MODBUS RTU requires to use the item.

Parameter:

IP_address - address of the station being a slave.

Declaration of Startup Time

Establishing connections with slaves is executed on the startup stage of the driver. The default time of startup duration is equal to 3 seconds. It may be modified by means of the item:

☑ ***STARTUP_TIME = number***

Meaning - startup time of the driver, during which connections are established with slaves.

Default value - 3 s.

Defining - manual.

☑ ***RECV_TIMEOUT=client_IP_address, number***

Meaning - for each slave type device a maximal time, which may elapse between sending a query and receiving an answer (so called receiving timeout) is determined. After exceeding the timeout the

connection is broken (and re-established). The timeout value is determined individually for each slave type device.

Default value - in case of slave type devices, for which any timeout declaration is not defined, it is assumed 5 seconds by default.

Defining - manual.

Parameter:

client_IP_address - IP address of a slave type device,

number - timeout value expressed in seconds.

LOG_FILE=file_name

Meaning - the item allows to define a file to which all diagnostic messages of the driver generated in the MASTER mode and the information about the contents of telegrams sent/received in this mode by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.

Default value - by default, the log file is not created.

Defining - manual.

Parameter:

file_name - log file name.

LOG_OF_TELEGRAMS =YES|NO

Meaning - the item allows to write to the log file (declared by means of the item LOG_FILE) the contents of telegrams sent between the driver operating in the MASTER mode and slave type devices. Writing the contents of telegrams should be used only while the **asix** start-up.

Default value - by default, the driver does not write the contents of telegrams to the log file.

Defining - manual.

LOG_FILE_SIZE=number

Meaning - the item allows to specify a log file size.

Default value - by default, the log file size is equal to 1 MB.

Parameter:

number - log file size in MB.

2. List of Figures

Figure 1. Format FLOAT (IEEE 745). *7*
Figure 2. Format LONG. *7*
Figure 3. Format ULONG. *7*

1. MODBUS_TCPIP - DRIVER OF MODBUS_TCP/IP PROTOCOL FOR OPEN MODBUS/TCP MODE 3

1.1. DRIVER USE 3
1.2. DECLARATION OF TRANSMISSION CHANNEL 4
1.3. SLAVE DRIVER CONFIGURATION 4
1.4. DECLARATION OF TRANSMISSION CHANNEL 10
1.5. MASTER MODE CONFIGURATION 11

2. LIST OF FIGURES 13